```
from sklearn import datasets
```

```
iris = datasets.load_iris()
iris
```

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.4, 3.9, 1.7, 0.4],
        [4.6, 3.4, 1.4, 0.3],
        [5. , 3.4, 1.5, 0.2],
        [4.4, 2.9, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.1],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
        [5.7, 3.8, 1.7, 0.3],
        [5.1, 3.8, 1.5, 0.3],
        [5.4, 3.4, 1.7, 0.2],
        [5.1, 3.7, 1.5, 0.4],
        [4.6, 3.6, 1. , 0.2],
        [5.1, 3.3, 1.7, 0.5],
        [4.8, 3.4, 1.9, 0.2],
        [5. , 3. , 1.6, 0.2],
        [5. , 3.4, 1.6, 0.4],
        [5.2, 3.5, 1.5, 0.2],
        [5.2, 3.4, 1.4, 0.2],
        [4.7, 3.2, 1.6, 0.2],
        [4.8, 3.1, 1.6, 0.2],
        [5.4, 3.4, 1.5, 0.4],
        [5.2, 4.1, 1.5, 0.1],
        [5.5, 4.2, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.2],
        [5. , 3.2, 1.2, 0.2],
        [5.5, 3.5, 1.3, 0.2],
        [4.9, 3.6, 1.4, 0.1],
        [4.4, 3. , 1.3, 0.2],
        [5.1, 3.4, 1.5, 0.2],
        [5. , 3.5, 1.3, 0.3],
        [4.5, 2.3, 1.3, 0.3],
        [4.4, 3.2, 1.3, 0.2],
        [5. , 3.5, 1.6, 0.6],
        [5.1, 3.8, 1.9, 0.4],
        [4.8, 3. , 1.4, 0.3],
        [5.1, 3.8, 1.6, 0.2],
        [4.6, 3.2, 1.4, 0.2],
        [5.3, 3.7, 1.5, 0.2],
        [5. , 3.3, 1.4, 0.2],
        [7. , 3.2, 4.7, 1.4],
        [6.4, 3.2, 4.5, 1.5],
        [6.9, 3.1, 4.9, 1.5],
        [5.5, 2.3, 4. , 1.3],
        [6.5, 2.8, 4.6, 1.5],
        [5.7, 2.8, 4.5, 1.3],
```

```
       [6.3, 3.3, 4.7, 1.6],
       [4.9, 2.4, 3.3, 1. ],
```

```
x = iris['data']
x
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
       [6.3, 3.3, 4.7, 1.6],
```

```
[4.9, 2.4, 3.3, 1. ]
```

```
y=iris['target']
y
#input columns
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)

#seperate training data and testing data
```

```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(120, 4)
(30, 4)
(120,)
(30,)
```

```
import tensorflow
from tensorflow import keras
from keras import Sequential
from keras.layers import Dense
from keras.optimizers import Adam

#when want to train model sequentially then we use Sequential class
#Dense class is used to establish fully connected network
```

```
model = Sequential()
model.add(Dense(8, activation='relu', input_dim=4))
model.add(Dense(6, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

```
/opt/conda/lib/python3.10/site-packages/keras/src/layers/core/dense.py:87: UserWarning: Do r
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
model.summary()

#see model summary
```

Model: "sequential_8"

Layer (type)	Output Shape	Param #
dense_24 (Dense)	(None, 8)	40
dense_25 (Dense)	(None, 6)	54
dense_26 (Dense)	(None, 3)	21

Total params: 115 (460.00 B)
Trainable params: 115 (460.00 B)
Non-trainable params: 0 (0.00 B)

```
model.compile(optimizer=Adam(learning_rate = 0.001), loss='sparse_categorical_crossentropy', me
# my traget label is not one hot encoded so that I use sparse_categorical_carossentropy
```

```
history=model.fit(x_train,y_train,epochs=50,batch_size = 16, validation_split=0.2)

#model training and store model in history
```

```
Epoch 1/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 2s 97ms/step - accuracy: 0.3707 - loss: 1.2982 - val_accuracy: 0.
Epoch 2/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.3296 - loss: 1.2893 - val_accuracy: 0.4
Epoch 3/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.2537 - loss: 1.3277 - val_accuracy: 0.4
Epoch 4/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.2899 - loss: 1.1770 - val_accuracy: 0.4
Epoch 5/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.3146 - loss: 1.1234 - val_accuracy: 0.4
Epoch 6/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.3187 - loss: 1.0753 - val_accuracy: 0.4
Epoch 7/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.3760 - loss: 0.9883 - val_accuracy: 0.4
Epoch 8/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.3115 - loss: 1.0115 - val_accuracy: 0.4
Epoch 9/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.2442 - loss: 1.0514 - val_accuracy: 0.4
Epoch 10/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.2763 - loss: 0.9981 - val_accuracy: 0.4
Epoch 11/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.3287 - loss: 0.9709 - val_accuracy: 0.6
Epoch 12/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6140 - loss: 0.9366 - val_accuracy: 0.7
Epoch 13/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6330 - loss: 0.9599 - val_accuracy: 0.8
Epoch 14/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6167 - loss: 0.9362 - val_accuracy: 0.8
Epoch 15/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6280 - loss: 0.9293 - val_accuracy: 0.8
Epoch 16/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6731 - loss: 0.8856 - val_accuracy: 0.8
Epoch 17/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5997 - loss: 0.9144 - val_accuracy: 0.8
Epoch 18/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5900 - loss: 0.8899 - val_accuracy: 0.8
Epoch 19/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6732 - loss: 0.8386 - val_accuracy: 0.8
Epoch 20/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6287 - loss: 0.8352 - val_accuracy: 0.8
Epoch 21/50
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6509 - loss: 0.8103 - val_accuracy: 0.8
Epoch 22/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6088 - loss: 0.8285 - val_accuracy: 0.8
Epoch 23/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6260 - loss: 0.8216 - val_accuracy: 0.8
Epoch 24/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6406 - loss: 0.7919 - val_accuracy: 0.8
Epoch 25/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6629 - loss: 0.7812 - val_accuracy: 0.8
Epoch 26/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6167 - loss: 0.7784 - val_accuracy: 0.8
Epoch 27/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6545 - loss: 0.7475 - val_accuracy: 0.8
Epoch 28/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6710 - loss: 0.7435 - val_accuracy: 0.8
Epoch 29/50
```

```
y_predict=model.predict(x_test)
y_predict

#model testing
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 162ms/step
array([[0.037067  , 0.29631764, 0.66661537],
       [0.08197067, 0.4568053 , 0.46122414],
       [0.70009696, 0.21160768, 0.08829538],
       [0.02927739, 0.29851007, 0.67221254],
       [0.05414385, 0.40039203, 0.5454642 ],
       [0.08182162, 0.46380627, 0.45437214],
       [0.724522  , 0.19626713, 0.0792108 ],
       [0.7935461 , 0.15297239, 0.05348155],
       [0.10064615, 0.46586555, 0.43348828],
       [0.07658035, 0.48460442, 0.43881518],
       [0.13420014, 0.49049935, 0.3753005 ],
       [0.04293352, 0.3362556 , 0.6208109 ],
       [0.02415861, 0.3004557 , 0.6753857 ],
       [0.10844847, 0.46430346, 0.42724806],
       [0.08375046, 0.49016234, 0.42608723],
       [0.7535415 , 0.17794782, 0.0685107 ],
       [0.17953767, 0.4716131 , 0.34884924],
       [0.0145287 , 0.3312519 , 0.65421945],
       [0.01933306, 0.37692726, 0.6037397 ],
       [0.03370807, 0.43746784, 0.5288241 ],
       [0.11302123, 0.44978428, 0.43719447],
       [0.11573584, 0.49492306, 0.3893411 ],
       [0.73639435, 0.18847556, 0.07513009],
       [0.69868577, 0.21257   , 0.08874425],
       [0.752589  , 0.1785875 , 0.06882355],
       [0.02227131, 0.38157305, 0.5961556 ],
       [0.7631718 , 0.17200169, 0.06482649],
       [0.11803571, 0.43420872, 0.44775555],
       [0.7511125 , 0.17951821, 0.06936932],
       [0.03899674, 0.34116375, 0.61983955]], dtype=float32)
```

```
y_test
```

```
array([2, 1, 0, 2, 2, 1, 0, 0, 1, 1, 1, 2, 2, 1, 1, 0, 1, 2, 2, 2, 1, 1,
       0, 0, 0, 2, 0, 1, 0, 2])
```

To improve performance,

1. Hyperparameter Tuning:

- Learning Rate: Experiment with slightly lower or higher learning rates (e.g., 0.0005 or 0.005).
- Batch Size: Try different batch sizes (e.g., 8, 32) to see how they impact training stability and convergence.

2. Increase Model Complexity:

- Add More Layers: Introduce additional hidden layers or increase the number of neurons in existing layers.

3. Regularization Techniques:

- Dropout: Add dropout layers to prevent overfitting.
- L2 Regularization: Apply L2 regularization to the dense layers.

4. Learning Rate Schedulers:

- Reduce Learning Rate on Plateau: Automatically reduce the learning rate when a metric has stopped improving.

5. Data Augmentation:

- Synthetic Data Generation: Although not typically needed for small datasets like Iris, you could augment the data using slight variations if the dataset were larger or more complex.

6. Cross-Validation:

- K-Fold Cross-Validation: Use cross-validation instead of a single train-validation split to better assess model performance.

7. Early Stopping:

- Stop Training Early: Use early stopping to halt training when validation performance no longer improves.

8. Feature Engineering:

- Standardization: Ensure that input features are standardized (mean=0, std=1) to help the model converge faster.

9. Optimize the Network Architecture:

- Experiment with Different Activations: Try other activation functions like LeakyReLU or ELU in hidden layers.

10. Use a Different Optimizer:

- Try Alternative Optimizers: Experiment with optimizers like RMSprop or Adam with different settings.

Don't run below shell it's just interpretatio of how you can do above task

2

```
model.add(Dense(10, activation='relu'))
```

3

```
#dropout

from tensorflow.keras.layers import Dropout
model.add(Dropout(0.3))  # 30% dropout rate

#L2-regularization

from tensorflow.keras.regularizers import l2
model.add(Dense(8, activation='relu', kernel_regularizer=l2(0.01)))
```

4

```
from tensorflow.keras.callbacks import ReduceLROnPlateau
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5)
model.fit(x_train, y_train, epochs=50, batch_size=16, validation_split=0.2, callbacks=[lr_sched
```

6

```
from sklearn.model_selection import KFold
kfold = KFold(n_splits=5, shuffle=True)
```

7

```
from tensorflow.keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
model.fit(x_train, y_train, epochs=50, batch_size=16, validation_split=0.2, callbacks=[early_st
```

8

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
```

9

```
from tensorflow.keras.layers import LeakyReLU
model.add(Dense(8))
model.add(LeakyReLU(alpha=0.1))
```

10

```
from tensorflow.keras.optimizers import RMSprop
model.compile(optimizer=RMSprop(learning_rate=0.001), loss='sparse_categorical_crossentropy', m
```

B: Image Data

- Neural Network implementation on mnist dataset (Image as an input)
- Task 1: Creating First Artificial Neural Network (ANN) using Keras and Tensorflow.

> Dataset: MNIST

- Task 2: Improve the performance of Artificial Neural Network.

```
from sklearn.datasets import load_digits
mnist = load_digits()
```

```
mnist
```

```
{'data': array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
        [ 0.,  0.,  0., ..., 10.,  0.,  0.],
        [ 0.,  0.,  0., ..., 16.,  9.,  0.],
        ...,
        [ 0.,  0.,  1., ...,  6.,  0.,  0.],
        [ 0.,  0.,  2., ..., 12.,  0.,  0.],
        [ 0.,  0., 10., ..., 12.,  1.,  0.]]),
 'target': array([0, 1, 2, ..., 8, 9, 8]),
 'frame': None,
 'feature_names': ['pixel_0_0',
  'pixel_0_1',
  'pixel_0_2',
  'pixel_0_3',
  'pixel_0_4',
  'pixel_0_5',
  'pixel_0_6',
  'pixel_0_7',
  'pixel_1_0',
  'pixel_1_1',
  'pixel_1_2',
  'pixel_1_3',
  'pixel_1_4',
  'pixel_1_5',
  'pixel_1_6',
  'pixel_1_7',
```

```
'pixel_2_0',
'pixel_2_1',
'pixel_2_2',
'pixel_2_3',
'pixel_2_4',
'pixel_2_5',
'pixel_2_6',
'pixel_2_7',
'pixel_3_0',
'pixel_3_1',
'pixel_3_2',
'pixel_3_3',
'pixel_3_4',
'pixel_3_5',
'pixel_3_6',
'pixel_3_7',
'pixel_4_0',
'pixel_4_1',
'pixel_4_2',
'pixel_4_3',
'pixel_4_4',
'pixel_4_5',
'pixel_4_6',
'pixel_4_7',
'pixel_5_0',
'pixel_5_1',
'pixel_5_2',
'pixel_5_3',
'pixel_5_4',
'pixel_5_5',
'pixel_5_6',
'pixel_5_7',
```

```
x = mnist['data']
x
```

```
array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 10.,  0.,  0.],
       [ 0.,  0.,  0., ..., 16.,  9.,  0.],
       ...,
       [ 0.,  0.,  1., ...,  6.,  0.,  0.],
       [ 0.,  0.,  2., ..., 12.,  0.,  0.],
       [ 0.,  0., 10., ..., 12.,  1.,  0.]])
```

```
y = mnist['target']
y
```

```
array([0, 1, 2, ..., 8, 9, 8])
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test =  train_test_split(x,y,test_size = 0.2, random_state = 42)
```

```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(1437, 64)
(1437,)
```

```
(360, 64)
(360,)
```

```
x_train = x_train / 255.0
x_test = x_test / 255.0
```

```
x_train[0].shape
```

```
(64,)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
```

```
model = Sequential()
model.add(Flatten(input_shape=(64,)))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

```
model.summary()
```

Model: "sequential_7"

Layer (type)	Output Shape	Param #
flatten_6 (Flatten)	(None, 64)	0
dense_21 (Dense)	(None, 64)	4,160
dense_22 (Dense)	(None, 32)	2,080
dense_23 (Dense)	(None, 10)	330

Total params: 6,570 (25.66 KB)
Trainable params: 6,570 (25.66 KB)
Non-trainable params: 0 (0.00 B)

```
model.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy', metr:
```

```
history = model.fit(x_train, y_train, epochs=50, batch_size=32, validation_split=0.2)
```

```
Epoch 1/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9277 - loss: 0.3128 - val_accuracy: 0
Epoch 2/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9386 - loss: 0.2734 - val_accuracy: 0
Epoch 3/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9413 - loss: 0.2679 - val_accuracy: 0
Epoch 4/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9341 - loss: 0.2528 - val_accuracy: 0
Epoch 5/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9469 - loss: 0.2252 - val_accuracy: 0
Epoch 6/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9436 - loss: 0.2299 - val_accuracy: 0
Epoch 7/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9335 - loss: 0.2403 - val_accuracy: 0
Epoch 8/50
```

```
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9425 - loss: 0.2259 - val_accuracy: 0
Epoch 9/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9514 - loss: 0.2041 - val_accuracy: 0
Epoch 10/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9584 - loss: 0.1839 - val_accuracy: 0
Epoch 11/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9472 - loss: 0.2083 - val_accuracy: 0
Epoch 12/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9487 - loss: 0.1845 - val_accuracy: 0
Epoch 13/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9593 - loss: 0.1676 - val_accuracy: 0
Epoch 14/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9591 - loss: 0.1725 - val_accuracy: 0
Epoch 15/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9553 - loss: 0.1651 - val_accuracy: 0
Epoch 16/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9531 - loss: 0.1670 - val_accuracy: 0
Epoch 17/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9604 - loss: 0.1565 - val_accuracy: 0
Epoch 18/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9486 - loss: 0.1606 - val_accuracy: 0
Epoch 19/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9636 - loss: 0.1432 - val_accuracy: 0
Epoch 20/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9673 - loss: 0.1427 - val_accuracy: 0
Epoch 21/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9649 - loss: 0.1457 - val_accuracy: 0
Epoch 22/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9630 - loss: 0.1519 - val_accuracy: 0
Epoch 23/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9608 - loss: 0.1552 - val_accuracy: 0
Epoch 24/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9571 - loss: 0.1391 - val_accuracy: 0
Epoch 25/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9689 - loss: 0.1205 - val_accuracy: 0
Epoch 26/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9712 - loss: 0.1181 - val_accuracy: 0
Epoch 27/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9643 - loss: 0.1303 - val_accuracy: 0
Epoch 28/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9678 - loss: 0.1191 - val_accuracy: 0
Epoch 29/50
```

- We can use same techniques as before for improve performance